

Pre-CLTK-Workshop

Digitale Sprachverarbeitung für historische Disziplinen

Sarah Lang

Nov.2019

Zentrum für Informationsmodellierung, Graz

Intro

Python-Installation

Erste Schritte in Jupyter / NLTK-Installation

Theorie

Intro to NLP concepts

NLP-Pipeline

Tools und Visualisierung

Grundbegriffe der Programmierung

Natural Language Processing in Python

CLTK Pipeline

Intro

1. Wir wollen die Anaconda Distribution installieren.
2. Wer hat das schon? Wer braucht das noch?
3. Welche Probleme gab es?
4. Danach müssen noch NLTK und CLTK installiert werden (dauert einen Moment!)

Linux-Bug

(ggf. Fehlermeldung 'conda not found')

Support:Conda not found error ● Support Navigator Issues → dann:

```
export PATH=~/.anaconda3/bin:$PATH ..... Path fehlt
conda --version ..... check if working
conda init ..... initialize
anaconda-navigator ..... launch navigator
```

When started: Click Jupyter Notebook. Choose directory. New → Python 3.

Erste Schritte in Jupyter / NLTK-Installation i

```
# press CTRL+ENTER to evaluate in Jupyter
# mit '#' markiert man Kommentare,
# also Infos für sich selbst, die der Computer aber ignorieren soll
1 * 4 + 5 / 8
# wie funktionieren die Klammern??

# Grundlegende Funktionalitäten sind bereits in der
# Standard-Bibliothek enthalten. Für Spezielleres
# müssen wir sog. 'Pakete' hineinladen.
# Das geht z.B. so:
import nltk
nltk.download() # Der Schritt ist nur 1x nötig,
# wenn das halt noch nicht installiert ist oder
# out-of-date oder man nicht alles gedownloaded
# hat beim letzten Mal und jetzt noch andere Sachen
# dazunehmen will (Paket ist relativ groß,
# daher könnte es von Interesse sein, nur selektiv manche Sachen zu ne
```

Erste Schritte in Jupyter / NLTK-Installation ii

```
# nicht vergessen im Prompt/Dialog zu klicken,  
# was man will (Progressbar startet), sonst passiert nix
```

```
from nltk.book import *  
text1 # Moby Dick
```

Theorie

Begrifflichkeiten

Quantitative methods in literary studies began in the sub-field of Stylometry: L.A. Sherman *Analytics of Literature* (1893). Now, there are many sub-fields and different names for quantitative methods for texts and language processing¹ in/for the Humanities:

Quantitative Text Analysis (QTA)

Computational Literary Studies (CLS)

Computational Linguistics

Text Mining

Franco Moretti's *Distant Reading*

Matthew Jocker's *Macroanalysis*

¹A fascinating introductory read on the 'statistical language model', underlying computerized language processing, is Wu 2018, Don't be put off by the emphasis on mathematics, it's really a series of easy-to-understand blog posts about language processing.

1. **exploratory**

Exploring a corpus for potentially interesting patterns. Be pointed to interesting phenomena; find new research questions; explore the 'potentials' of the text/corpus.

2. **descriptive**

Describing phenomena quantitatively which we already 'feel' are there but we want to grasp them more 'objectively'. Confirm 'qualitative hypotheses quantitatively.

3. → both are not 'results'! They still require interpretation to be valid for Humanities purposes.

Intro to NLP concepts

Bag-of-words (BOW)

type vs. token

To be or not to be. = 6 *tokens*, 4 *types*. → aus den *types* wird der *bag of words* aufgebaut, darin werden dann alle Vorkommnisse (*tokens*) gezählt.

type

beschreibendes Kriterium

token

Analyseeinheit

case-folding

alles in *lowercase* analysieren: hat vor und Nachteile (z.B. Eigennamen), am Satzanfang evtl. sinnvoll, außer für Stilometrie.

Stattdessen evtl. *truecasing*

Je nach Entscheidung bzgl. *case-folding/truecasing* ist ein Wort in Großschreibung ein anderer *type* als in Kleinschreibung!

Was geht dabei alles verloren?

- Wortstellung
- Zusammenhang
- Phrasen
- Reihenfolge
- Ironie, Sarkasmus, Negation

Zipf'sche Verteilung (*Zipf's Law*)

starker (proportionaler) Abfall der Häufigkeit (und absoluten Streuung) mit zunehmendem Rang

“ Zipf's law states that given a large sample of words used, the frequency of any word is inversely proportional to its rank in the frequency table.
(Wikipedia) ”

Stopwords

überproportional häufige Wörter (und, *the*, etc.). In Listen gesammelt, meist aus der Analyse entfernt, da sie die interessanten Partien 'verdecken'. *To be or not to be* = alles Stopwords.

Wortfrequenzanalysen

siehe 'Statistische Grundlagen'

Unigramme vs. n-Gramme

Unigram: Jeglicher Kontext geht verloren.

n-Gramme: *sparse data-Problem*

d.h. wir haben fast nie genug Daten dafür, außer bei *sehr* großen Korpora. Die Häufigkeiten werden dabei schnell so 'selten', dass nichts statistisch Relevantes mehr dabei ist. Mehr als 3-5-Gramme sind daher nicht üblich. (zumal für historische Sprachen ohnehin viel zu kleine Korpora erhalten sind!)

KWIC (=Keyword in Context)

Kollokation

Konkordanz

“

Unter Konkordanz (zurückgehend auf lat. *concordare* „übereinstimmen“), versteht man in den Textwissenschaften traditionellerweise eine alphabetisch geordnete Liste der wichtigsten Wörter und Phrasen, die in einem schriftlichen Werk verwendet werden. Der Begriff stammt aus der Bibelwissenschaft. [...]

Konkordanzen sind heute in der Regel elektronisch erstellte Trefferlisten, die sich aus der Suche meist nach einem Wort oder einer Phrase, eigentlich aber aus der Suche nach jeder beliebig definierbaren Zeichenkette ergeben. In einer Konkordanz ist meistens auch die nächste sprachliche Umgebung des gesuchten Ausdrucks, der sogenannte *Kontext* angeführt, also beispielsweise der gesamte Satz, in dem ein gesuchtes Wort auftritt.

Als Synonyme für Konkordanz gelten fallweise die Ausdrücke *Register* und *Index* oder *Index verborum* („Verzeichnis der Wörter“).

In der Korpus- und in der Computerlinguistik haben sich zudem, auch im Deutschen, der Ausdruck *Key Word in Context* sowie dessen Abkürzung *KWIC* als Benennungen für den in einer Konkordanz angezeigten Suchbegriff eingebürgert.

(Wikipedia) ”

Sentiment Analysis

Eine weitere bekannt gewordene 'Wörterbuch-Methode'. Man macht quasi ganz normal die Auszählung und verbindet dann die Tabelle per *join* mit einem *sentiment dictionary*, das Wörter in Kategorien bewertet. Allerdings passieren natürlich Missverständnisse: Bsp: Jane Austens häufigstes Nicht-Stopwort ist 'Miss', aber in *lowercase* (da *case-folding*) wird es natürlich fehlinterpretiert als 'vermissen' und damit als negatives Gefühl gewertet!

Hintergrundinfos zu **statistischen Grundlagen**
und **Korpusaufbau** siehe Cheatsheet!

Bitte ggf. selbst aneignen / einlesen.

“ Reading ‘more’ seems hardly to be the solution. Especially because we’ve just started rediscovering what Margaret Cohen calls the ‘great unread’. [Apart from “its canonical fraction, which is not even 1 per cent of published literature”] there are [thousands of books] – no one really knows, no one has read them, no one ever will.² ”

Franco Morettis *Distant Reading*

Vorstellung serielles Lesen vs. menschliches *Close Reading*.

Behauptet, Literaturwissenschaft wäre nicht mehr vollständig ohne quantitative Aspekte. Das serielle Lesen aber auch irgendwie ‘als Alternative’ anstatt *close reading*.

“ [...] you *reduce* a text to a few elements, and *abstract* them from the narrative flow, and construct a new, *artificial* object. [...] And with a little luck, these maps will [...] possess ‘emerging’ qualities, which were not visible at the lower level. [...] Not that the map itself is an explanation, of course. It offers a model [...]”³

Matthew Jockers's *Macroanalysis*

Vorstellung einer Zoom-Bewegung: Die quantitative Analyse bietet Anstöße für das *Close Reading*, etc. Er fordert die Unterscheidung zwischen *reading* und *analysis* – der Überblick 'aus der Ferne' ist für ihn nicht 'Lesen', wie etwa Morettis Benennung suggerieren würde.

1. Kontextualisierung durch das *zooming out*
2. andererseits aber auch extremes *close reading*: So viel Details wie dem Computer können einem Menschen fast gar nicht auffallen, weil wir Details ja gar nicht so richtig wahrnehmen.
3. besser informiertes Verstehen der Primärtexte: die *macroscale* gibt weniger 'anekdotische' Beweise als das sehr genaue Lesen nur eines einzigen Texts.
4. 'harvesting findings, not facts'

‘Mixed Methods’-Ansatz, Wechselspiel

“ This is not close reading; this is macroanalysis, and the strength of the approach is that it allows for both zooming in and zooming out.⁴ ”

Pipers cultural analytics

²Moretti 2013, 45.

³Moretti 2005, 53.

⁴Jockers 2013, 23.

NLP-Pipeline

(Pre-Processing)

→ fällt in unserem Fall normalerweise unter das sog. (Pre-Processing)

Tokenizer

Einfachste Tokenizer: Nur der Whitespace/Leerzeichen ist Trenner.
Erweiterbar durch Hinzufügen von Punctuation oder sprachinternen Spezifika.

Sprachgebunden, z.B. frz. *l'enfant*, lat. *dixitque*, en. *don't* auflösen, ...

Wie wird mit **compounds** (zusammengesetzten Wörtern) umgegangen? Fremdsprachliche **Lehnwörter**, die zusätzlich noch stehende Wendungen sind (z.B. *en masse*)?

Geht meist ohne Wörterbuch – für die meisten anderen NLP-Methoden müssen Wörterbücher und Grammatiken vorliegen (!

ist für einen historischen Sprachstatus absolut nicht selbstverständlich) → Computer kann nicht denken und ist z.B. mit der uneinheitlichen Orthographie von frühneuzeitlichem Deutsch völlig überfordert: Methoden davon teilweise in die völlige Impraktikabilität reduziert!

→ aber geht natürlich super für moderne lebendige Sprachen, v.a. Englisch.

“ **But what is a word?** We tend to think of a word as a unit of meaning that often has an analog in the real world. [...] A computer doesn't know what a word is and certainly has no sense of what words might refer to. [...] Words are usually bounded by spaces and punctuation, and a computer can be told to split a long string (text) into shorter strings (words) by looking for the demarcation characters [...] – a process called tokenization.⁵ ”

⁵Sinclair und Rockwell 2016, 283.

Lemmatization

Mithilfe von **Wörterbuch/Vokabular, Grammatik und morphologischer Analyse** der Wörter die Grundform (das *Lemma*, wie man es im Wörterbuch hat) zu finden, damit alle Wortformen korrekt als Formen desselben *type* gezählt werden können.

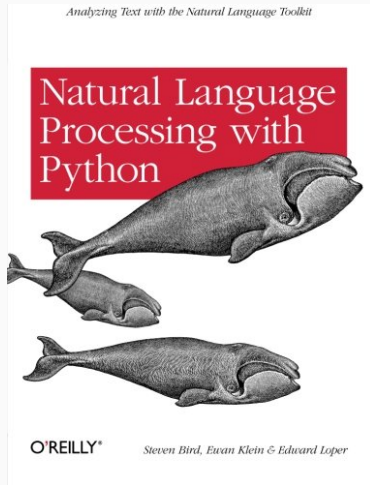
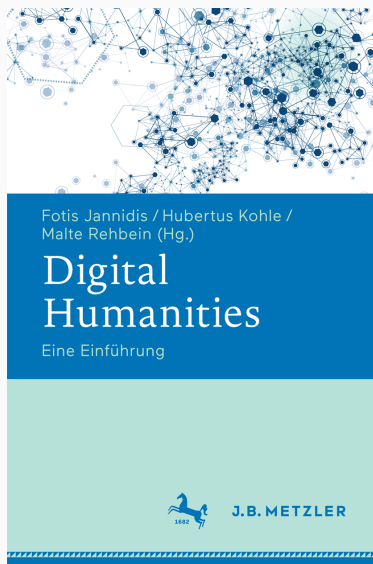
Stemming

Wortende bis zur Wurzel abschneiden, nicht unbedingt auf linguistisch korrekte Art und Weise.

Parsing

POS tagging

Buch-Tipps zum Einstieg



Tools und Visualisierung

Siehe Kapitel 23 'Informationsvisualisierung' (DH-Einführungsbuch)⁶

Grafische Elemente als Kommunikationsmittel ● visuelle statistische Darstellungen als Repräsentationen von Sprache ● Entlastung des Geistes bei großen Datenmengen: *snapshot* statt Gesamtwerk. ● 'method for seeing the unseen' ● durch die Visualisierung werden abstrakte Daten räumlich angeordnet ● oft interaktive Schnittstellen, um den Denkprozess zu begleiten

Funktionen

1. **anschauliche Präsentation** von Daten/Ergebnissen, am Ende des Forschungsprozesses
2. **konfirmative Analyse:** Sichweisen (*views*) auf Daten erzeugen, die erlauben, Hypothesen zu verifizieren oder falsifizieren. Teil des Forschungsprozesses.
3. **explorative Analyse:** nicht hypothesengetrieben, Strukturen oder Trends erkennen. → interaktive Werkzeuge, zu Beginn des Forschungsprozesses
4. **Visualisierung** (z.B. Storytelling)

makros vs. mikro

Einerseits Aufzeigen übergeordneter Strukturen ('distant reading'),
aber auch Detailblick in die Daten.

Vorgehen

raw data → (Vorverarbeitung) → *data tables* → *visual structures* →
views.

→ Tufte 1983: *A silly theory means a silly graphics*.

⁶vgl. Rehbein 2017.

“ *A fool with a tool is still a fool!* (Grady Booch) ”

Frage nach der Aussagekraft der Daten

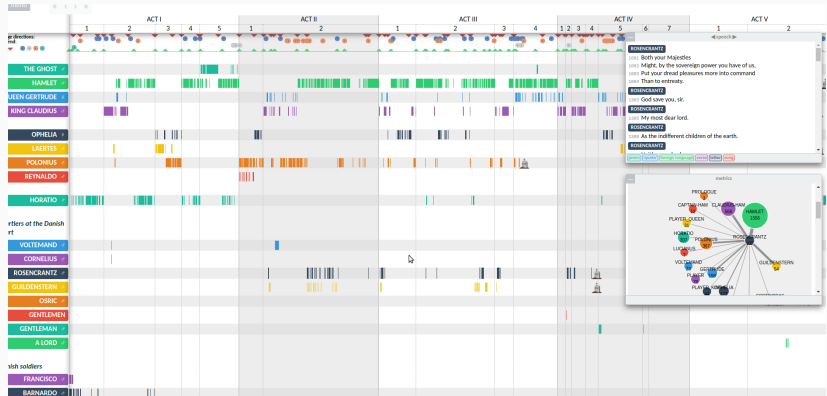
“ Nur aus der bloßen Existenz von Daten kann nicht auf deren Bedeutsamkeit geschlossen werden, und nicht alles, was (in einem bestimmten Kontext) bedeutsam ist, hinterlässt messbare Datenspuren.

Ein weiterer Schritt ist durch den Forscher selbst zu vollziehen: Visualisierungen sind häufig gut geeignet, um Strukturen und Muster zu zeigen. Sie liefern allein für sich aber keine Erklärungen. Mit anderen Worten: Korrelationen oder Koinzidenzen, wie sie sich vielleicht in den Daten erkennen lassen, bedeuten noch keine Kausalität, eine Wiederkehr noch keine Gesetzmäßigkeit. Erklärungen lassen sich erst durch Interpretation der Visualisierungen, oft unter Bezugnahme von anderen Daten,

Quellen oder auch Methoden, durch den Forscher folgern.
(*Rehbein 2017, 341–341*) ”

Rolling Window

To See Or Not to See: Shakespeare-Visualisierung ● Annotationsbasiert



WordTree

word tree

Fourkondeln

 Springer

one pd

one pd

Frankenstein

exclaimed

- he, "how glad I am to see you! How fortunate that you should be here at the very moment of my alighting!"

- Henry, when he perceived me weep with bitterness, "are you always to be unhappy? My dear friend, what has happened?"

continued he, stopping short and gazing full in my face, "I did not before remark how very ill you appear; so thin and pale; you look as if you had been watching for several nights."

had at first been silent, and indeed appeared hardly to have force enough to attend, now roused himself; his eyes sparkled, and his cheeks flushed with momentary vigour.

— was dozing, awoke and asked the cause of the tumult. "They shout," I said, "because they will soon return to England."

thy aunt, in her last illness, with the greatest affection and care and afterwards attended her own mother during a tedious illness, in a manner that excited the admiration of all who knew her, after which
 your son, your kinsman, your early, much-loved friend; he who would spend each vital drop of blood for your sakes, who has no thought nor sense of joy except as it is mirrored also in your dear countenance;
 be not equitable to every other and trample upon me alone, to whom thy justice, and even thy clemency and affection, is most due.

I was benevolent; my soul glowed with love and humanity; but am I not alone, miserably alone? You, my creator, abhor me; what hope can I gather from your fellow creatures, who owe their miserable state only to my sins, to your mercy and goodness?—wretched, wretched, and lowly, wherefore are you desponding and sorrowful?" In truth, I was occupied by gloomy thoughts and neither saw the descent of the evening star nor the golden sunrise reflected in the Rhine.

you belong then to my enemy—to him towards whom I have sworn eternal revenge; you shall be my first victim.

- Generous and self-devoted being! What does it avail that I now ask thee to pardon me? I, who irretrievably destroyed thee by destroying all thou lovedst

- If thou wert yet alive and yet cherished a desire of revenge against me, it would be better satiated in my life than in my destruction.

or, the Modern Prometheus by Mary Wollstonecraft (Godwin) Shelley CONTENTS Letter 1 Letter 2 Letter 3 Letter 4 Chapter 1 Chapter 2 Chapter 3 Chapter 4 Chapter 5 Chapter 6 Chapter 7 Chapter 8
my pity amounted to horror; I abhorred myself. But when I discovered that he, the author at once of my existence and of its unspeakable torments, dared to hope for happiness, that while he accused

more, far more, will I achieve; treading in the steps already marked, I will pioneer a new way, explore unknown powers, and unfold to the world the deepest mysteries of creation.

he will punish you. You dare not keep me." "Frankenstein! you belong then to my enemy—to him towards whom I have sworn eternal revenge; you shall be my first victim.

"Geneva, May 12th, 17—," Clerval, who had watched my countenance as I read this letter, was surprised to observe the despair that succe-

You accuse me of murder, and yet you would, with a satisfied conscience, destroy your own creature. Oh, praise the eternal justice of man!

it; an excellent quality in a young man. Young men should be diffident of themselves, you know, M.

calars of his creature's formation, but on this point he was impenetrable. "Are you mad, my friend?"

discovered that I made notes concerning his history; he asked to see them and then himself corrected and augmented them in many places, but principally in giving the life and spirit to the conversations he has daily declined in health; a feverish fire still glimmers in his eyes, but he is exhausted, and when suddenly roused to any exertion, he speedily sinks again into apparent lifelessness.

⁵ still lie. I must arise and examine. Good night, my sister. Great God! what a scene has just taken place!

¹ "would yet have lived." "And do you dream?" said the demon. "Do you think that I was then dead in a

⁵ had said of his powers of eloquence and persuasion, and when I again cast my eyes on the lifeless form of my friend, ind

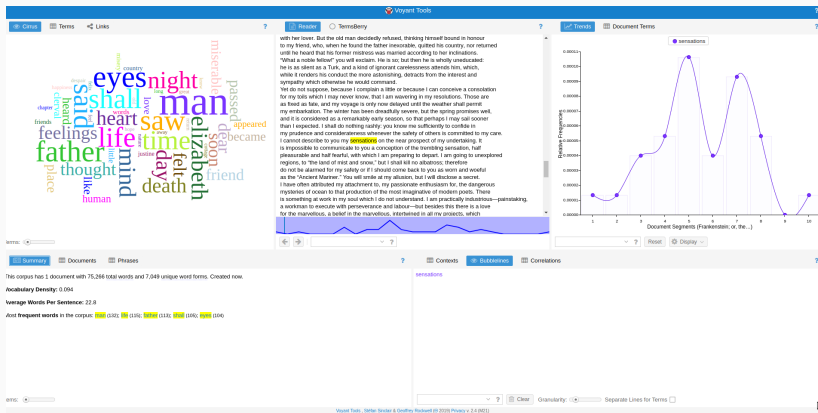
¹ your friend, seem to have a knowledge of my crimes and his misfortunes. But in the detail which he gave you of them he could not sum up the hours and m

, I perceived Henry Clerval, who, on seeing me, instantly sprung out. "My dear Frankenstein," exclaimed he, "how glad I am to see you! How fortunate that you should be here at the very moment of my alighting!"

nothing could equal my
delight on seeing Clerval;
his presence brought back
my thoughts my father,
my mother, my friends, those
scenes of home so dear to my
recollection. I grasped his
hand, and in a moment forgot
myself, and the whole scene
flashed suddenly, and for the
first time during many
months, calm and serene joy
possessed my mind. I was,
therefore, in the most
cordial manner, and we
walked towards my college
and my friends, and I was
for some time about my
mutual friends and his own
good fortune in being
restored to me. I said to
Ingloust: "You may easily
believe," said he, "how
great was the difficulty to
be overcome, and how much
necessary knowledge was not
comprised in the noble art
of book-keeping; and, indeed,
I am not at all surprised that
incredulous to the last, for
his constant answer was
unwearied entreaties was the
only means that the
schoolmaster in the Vicar of

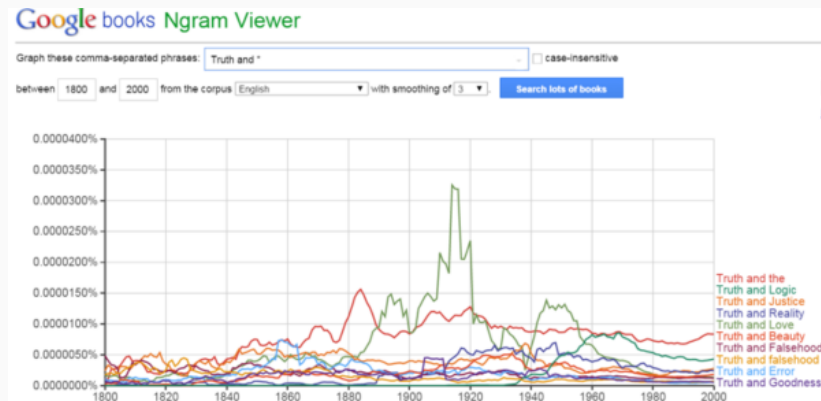
Voyant Tools

Blogpost zu Voyant ● features many standard functions for single and multiple documents ● XML input possible as well



Google Books N-Gram Viewer

Shai Opir: Bsp. zu 'Truth'



Stanford CoreNLP

Stanford CoreNLP ● Online-Tool für CoreNLP ● Einsteiger-Tutorial zu CoreNLPs Funktionen

NLP-Tools ii

Stanford CoreNLP 3.9.2 (updated 2018-11-29)

— Text to annotate —
"My dear Frankenstein," exclaimed Henry, when he perceived me weep with bitterness, "are you always to be unhappy? My dear friend, what has happened?"

— Annotations —
parts-of-speech x named entities x dependency parse x openie x

— Language —
English

Submit

Part-of-Speech:

1 " My dear Frankenstein , " exclaimed Henry , when he perceived me weep with bitterness , " are you always to be unhappy ?

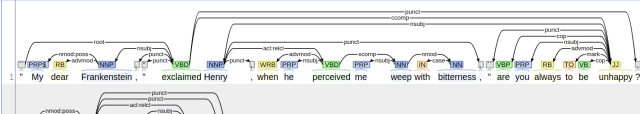
2 My dear friend , what has happened ? "

Named Entity Recognition:

1 " My dear Frankenstein , " exclaimed Henry , when he perceived me weep with bitterness , " are you always to be unhappy ?

2 My dear friend , what has happened ? "

Basic Dependencies:



Word and Phrase: Konkordanzen, Frequenzlisten, Texte analysieren
(ein ganzer Roman scheint ihm zu viel zu sein)

Konkordanz, Beziehungen zu anderen Wörtern, etc. Vom einen Begriff zum nächsten hangeln, ...

WordWanderer

[Options](#) [Help](#)

affection again agony air alone already although always anguish appear appearance appeared around arrived attention away beautiful
beauty became become before began beheld being believe beloved body brother brought called came care cause change chapter
child clerval cold come companion concerning continued cottage countenance country cousin creature creatures dark day days dead
death degree delight desire desired despair destroyed die died discovered door down during earth elizabeth end endeavoured endured
entered even evil existence expressed eyes family far father fear feel feeling feelings felix fellow felt few fiend filled find fire
first fixed food form found frankenstein free friend friends gave geneva gentle girl good great greater greatest grief ground hand
hands happiness happy hardly having head heard heart henry here himself home hope hopes horror hour hours house human ice
idea imagination indeed itself journey joy justine kind kindness knew know knowledge lake last lay leave left length letter life light
little live lived long longer looked lost love loved lovely made make man manner many means men mind mine miserable
misery moment monster months more morning mother mountains much murderer myself native natural nature near nearly
never new night nothing now occupied old once one others out over part passed peace perceived place placed pleasure
poor possessed power present promise quickly quitted rage read remained replied resolved rest return returned room safe same sat saw
scene science sea see seek seemed sensations several shall sight sister sometimes soon soul spent spirit spirits spoke state still
strange stranger such suddenly sun sweet take taken tale tears therefore think thought thoughts thousand through thus time took
towards town turned two under until up upon various very victor voice well whole whose william wind wish within without woman
wood words work world wretch wretched years young yourself "i "the "

Select a word to see its context or drag a line between two words to compare them.

WordWanderer iii

[illegible]

... slave of passion ; and quelling the dark tyranny of	despair	, he led me again to converse concerning myself personally ...
... the world before you , and have no cause for	despair	. but i—I have lost everything and cannot begin life ...
... the reuss . but when he entered , misery and	despair	alone welcomed him . beaufort had saved but a very ...
... him with the greatest tenderness , but she saw with	despair	that their little fund was rapidly decreasing and that there ...
... void that presents itself to the soul , and the	despair	that is exhibited on the countenance . it is so ...
... i read this letter , was surprised to observe the	despair	that succeeded the joy i at first expressed on receiving ...
... ; my imagination was busy in scenes of evil and	despair	. i considered the being whom i had cast among ...
... desire , and represented caroline beaufort in an agony of	despair	, kneeling by the coffin of her dead father
... , but words cannot convey an idea of the heart-sickening	despair	that i then endured . the person to whom i ...
... i could conceal the horrid anguish that possessed me .	despair	! who dared talk of that ? the poor victim ...

Grundbegriffe der Programmierung

Communicating with computers i

Algorithm

Recipe providing a formalized way to automate carrying out a task.

can be in 'pseudo-code' ("Whenever you find a verb, do this. When you find a noun, do that." etc.)

Algorithmic thinking

The art of communicating effectively with a computer

Debugging

is the art of error correction There are different types of errors, most importantly:

Syntax errors

mean that you didn't respect the conventions of your programming language. ~Typos

Logic errors

~design flaw. Usually these are 'bigger' errors in the program flow compared to the more localized syntax errors (typos, missing semicolon or closing brackets, etc.).

Semantic errors

mean that the code runs alright, but it doesn't do what you intended it to do.

Communicating with computers ii

Runtime error

A program crashes due to unexpected input but the compiler can't know someone will input something which will cause the crash (such as an unexpected and thus uncaught division-by-zero error which you should actually prevent). There are many recurring types of errors which consequently have received their own names, such as index-out-of-range error or off-by-one error, to name to most well-known examples.

Error handling

Errors should be handled in your program flow, this means they should be anticipated, 'caught' or 'thrown'. An error message will specify what kind of problem happened which makes debugging easier. An expected and thus properly handled error prevents the program from crashing whenever something problematic happens (which shouldn't happen!). If you have an advanced error handling in place, your program will know how to react when different types of errors happen. However, this is maybe overkill for tiny script-like programs where a crash is not an issue.

Input and Output

Input and output (sometimes referred to as I/O) are base concepts of how computing

Communicating with computers iii

works: You give input to a function or computer program and expect it to return a certain output. In between, some processing happens.

Variable

A variable is a container for storing information. A variable is a container for storing information, kind of like in maths where a variable is a placeholder for a yet unknown value. It is named (such as 'x' in maths). Calling the name usually causes it to give you its 'contents'.

Function

A function is a generalized bit of code, made for general-purpose reuse. So essentially, you abstract what is being done. Instead of "process my document test.txt" you would write something like `process(text)` and would be able to pass a document of any name to this `process()` function like `process("test.txt")` or `process(unicorn.txt)`. This has the awesome advantage that it doesn't matter whether your document is called 'test' or 'unicorn' or, in fact, any other crazy name you can come up with. A function usually has a 'signature' that means a list of what parameters you pass to it (in the definition of a function, you say which type of data will be inputted) and what it will return.

Communicating with computers iv

Parameter

is a variable in the declaration of a function. Together with variables (and parameters *are* a type of variable), parameters contribute to the abstraction of functions. This is a good thing.

Argument

is almost the same as a parameter, but the difference is that it means the actual value which gets passed to the function. So when using a function, you can pass the number 3 to it. In the definition/declaration, you say that an integer variable will be passed to it (=the parameter is an integer type value).

Return value

What a function will return as a result. This has a 'return type', specifying which data type will be returned. This can sometimes be good to know so that we don't get confused in further processing steps.

However, the point about the whole function-abstraction-thing is that we really shouldn't need to know exactly how the processing is done in a function. We only need to know what we put in (input) and what we expect to get out of it (output). This

Communicating with computers v

makes it easier to build really complex programs where you don't want to have to worry about how exactly things were implemented in the detailed single processing steps.

Library

A library (or package) is essentially nothing else than a bunch of functions, only that when you load them into your program, you can profit from (usually) really well-done and maintained functions somebody else made.

When you use code from a library, it means that this code is not part of the standard repertoire ('vocabulary') of your language: It is a set of functions made using this standard repertoire just as your own functions are. When you invoke or 'load' them, it's essentially as though the computer would just copy their text into your document behind the scenes (and this is exactly what happens behind the scenes).

Functions coming from libraries can have the name of their library prefixed as a 'namespace'. This can sometimes be a little confusing, especially for beginners because it obscures where which names belong to in longer expressions and requires you to know/remember from which libraries your used functions come.

Communicating with computers vi

Redundancy is always an unnecessary source of error. Avoid redundancy at all costs. Except for in data archiving. Here, redundancy is security.

Divide et impera

In the beginning, we learned that an algorithm (and thus, also it's concrete *implementation*, a program) is essentially a cooking recipe which allows us to automate things which are simple enough so they lend themselves to being automated. Sometimes, even things which look complicated at first can be reduced to simpler substeps. This is called the '**divide et impera**' principle in computer programming.

However, cooking recipes always expect the same input (the ingredients, so to speak). If the input differs from what's specified (even just a tiny little bit!), the computer really doesn't know what to do with it because the computer doesn't think and doesn't have common sense.

Loop

means repeating an action for a specified number of times. If something goes wrong in the definition of the ending condition (which always need to be defined!), you can

end up with an 'infinite loop' which will cause the program to hang or 'freeze' and eventually crash. For-loops repeat for a fixed amount of times, while loops repeat while a condition is true, do-while-loops execute at least one time and then behave as while-loops.

Conditional

is an if-then-type decision which can be used to regulate program flow. You can define multiple if cases, but also if-else and else cases which will handle everything which doesn't match the condition.

Suffice it to say, for now, that there are different ways of storing data. As a beginner, especially if you're using a loosely typed language such as Python, this isn't all that important at the beginning so I don't want to be too verbose as not to confuse you with irrelevant facts.

There are simple ('primitive') data types, such as integer, characters, floating point numbers or strings; as well as complex data types such as hashmaps, lists, dictionaries, etc. which are specific to the implementation in a programming language. There are also ways for you to define your own complex data types (such as defining

that if you want to store data on people, you need their names, ages and addresses and address always consists of X, Y and Z). Python is a 'dynamically (or weakly) typed' language, so you don't need to perform specific operations yourself (as opposed to 'strongly typed' languages like C) such as:

Type-casting

means to 'change' data from one type to another. A classic example is that '1' if typed into a terminal, actually is a string, not a number to the computer. In order to calculate with it, you need to change it to an integer. But this is not an issue in Python, so don't worry about that now.

Script / Scripting

Scripting in a terminal is not the same thing as programming or writing software (which contains memory allocation, error handling, etc.).

“ To get a computer to do anything, you (or someone else) have to tell it exactly – in excruciating detail – what to do. Such a description of “what to do” is called a *program*, and *programming* is the activity of writing and testing such programs. [...] The difference between such [everyday] descriptions and programs is one of degree of precision: **humans tend to compensate for poor instructions by using common sense, but computers don’t.** [...] when you look at those simple [everyday] instructions, you’ll find the grammar sloppy and the **instructions incomplete.** A human easily compensates. [...] In contrast, computers are *really* dumb.⁷

We also [...] to assume “common sense”. **Unfortunately, common sense isn’t all that common among humans and is totally absent in computers** (though some really well-designed programs can imitate it in specific, well-understood cases).⁸ ”

⁷Stroustrup 2014, 44. Hervorhebung hinzugefügt.

⁸Stroustrup 2014, 35. Emphasis added.

First steps in Python i

```
# press CTRL+ENTER to evaluate in Jupyter
# mit '#' markiert man Kommentare,
# also Infos für sich selbst,
# die der Computer aber ignorieren soll
1 * 4 + 5 / 8
# wie funktionieren die Klammern??

# Grundlegende Funktionalitäten sind bereits
# in der Standard-Bibliothek enthalten.
# Für Spezielleres müssen wir sog. 'Pakete' hineinladen.
# Das geht z.B. so:
import nltk
nltk.download()
# Der Schritt ist nur 1x nötig, wenn das halt noch nicht
# installiert ist oder out-of-date oder man nicht alles
# gedownloaded hat beim letzten Mal und jetzt noch andere
# Sachen dazunehmen will (Paket ist relativ groß,
```

First steps in Python ii

```
# daher könnte es von Interesse sein,  
# nur selektiv manche Sachen zu nehmen)  
# nicht vergessen im Prompt/Dialog zu klicken,  
# was man will (Progressbar startet), sonst passiert nix
```

```
from nltk.book import *  
text1 # Moby Dick
```

Natural Language Processing in Python

Basics of NLTK (Natural Language Toolkit)

Natural Language versus Artificial Language (i.e. mathematical notations, etc.)

NumPy and Matplotlib have to be installed to generate plots! (should be included in Anaconda anyway)

“ [Programming/Learning Python-NLTK] is just like learning idiomatic expressions in a foreign language: you’re able to buy a nice pastry without first having learned the intricacies of question formation.⁹ ”

⁹Bird, Klein und Loper 2009, xii.

Things to get started i

```
import nltk
nltk.download()
from nltk.book import *
text1 # Moby Dick

text1.concordance("monstrous")
text1.similar("monstrous")
text2.similar("monstrous")
text2.common_contexts(["monstrous", "very"])
# import numpy, matplotlib
text4.dispersion_plot(["citizens", "democracy", "freedom", "duties", ""])
# NumPy and Matplotlib are needed to plot this

len(text3) # word count
sorted(set(text3)) # list unique words in sorted order
len(set(text3)) / len(text3) # number of word types / tokens = measure
text3.count("smote") # word count for 'smote'
```

Things to get started ii

```
100 * text4.count('a') / len(text4) # percentage of 'a' in text

def lexical_diversity(text): # define this as function
    return len(set(text)) / len(text)

def percentage(count, total): # has to define above of the first usage
    return 100 * count / total

lexical_diversity(text3)
percentage(4, 5)
percentage(text4.count('a'), len(text4))
```

Texts as lists of words i

```
sent1 = ['Call', 'me', 'Ishmael', '.'] # Moby Dick first sentence as w
# stored in a variable named 'sent1'
# attention: var names are case-sensitive, so sent1 is not the same as
# some theoretically possible names (like 'not') are reserved words an
sent2 # (if you get an error which says that sent2 is not defined, you
sent4 + sent1 # concatenation (adding strings / text)
sent1.append("Some")
sent1 # word has been appended
text4[173] # get element via numeric index
# the first index always starts with 0, thus the last is (len(thingy)
text4.index('awaken') # find out this index for a specific word
text5[16715:16735] # getting elements like this is called 'slicing'
```

Texts as lists of words i

```
name = 'Monty'
name[0] # a word/string can be processed just like a list
name[:4] # get first 4 characters
name + '!'
'.join(['Monty', 'Python']) # joined together with one whitespace as
'Monty Python'.split() # splitted into a list

# How can we automatically identify the words of a text that are most
fdist1 = FreqDist(text1) # frequency distribution
print(fdist1)
fdist1.most_common(50)
fdist1['whale']

Vocab = set(text1)
# get all the words from the vocabulary which are longer than 15 characters
long_words = [w for w in Vocab if len(w) > 15]
# w is an arbitrary shorthand for word
```

Texts as lists of words ii

```
# you could also just say 'word' but it's longer to type it  
sorted(long_words)
```

Python data types i

Interactive tutorials: Hour of Python

Lists: related things

Python offers a tool called lists to keep track of related 'things', or values.

```
grades = [97, 62, 85, 76, 99, 93]
grades[0] # get by index. output: 97
# (IndexError if index non-existent)
names = ['Anna', 'Bob']
container = [grades, names]
# nested lists using variables
an_empty_list = []
grades.append(42)
grades.insert(2, 'spam') # [97,62,'spam',85,76,99,93,42]
# extending with + or
grades.extend(['foo'])
```

```
del grades[0:2]
grades.pop(3)
grades.remove(85)
# will remove first occurrence of value
# (ValueError if non-existent)
# ['spam', 76, 93, 42, 'foo']
```


Python Dictionaries i

Dictionaries: key-value built-in to Python, for storage, to translate keys to values.

```
foods = {'a': 'apple', 'b': 'banana', 'c': 'cookie'}  
foods['d'] = 'dates'  
y = {'spam': 'eggs'}  
x.update(y) # merge y into x  
foods['d'] # dates  
# for the getter: KeyError if search by value  
del foods['d']
```

Texts as lists of words i

```
# [NLP book, Chap. 3.2] very long words are often hapaxes (i.e., unique)
sorted(w for w in set(text5) if len(w) > 7 and fdist1[w] > 7)
```

```
# [NLP book, Chap. 3.3] A collocation is a sequence of words that occurs together
list(bigrams(['more', 'is', 'said', 'than', 'done']))
text4.collocations()
text8.collocations()
```

```
fdist1.tabulate()
fdist1.plot()
```

```
sorted(w for w in set(text1) if w.endswith('ableness'))
sorted(term for term in set(text4) if 'gnt' in term)
[w.upper() for w in text1] # capitalize words
len(set(word.lower() for word in text1)) # case-folded list of words
```

Nested conditions i

```
if len(word) >= 5:
    print('word length is greater than or equal to 5')
for word in ['Call', 'me', 'Ishmael', '.']:
    print(word)

sent1 = ['Call', 'me', 'Ishmael', '.']
for xyzzzy in sent1:
    if xyzzzy.endswith('l'):
        print(xyzzzy)
        print(xyzzzy, end=' ') # print not as newline but space-separat
```

[http://docs.cltk.org/en/latest/
installation.html](http://docs.cltk.org/en/latest/installation.html) → CLTK-Installier-Spaß,
juhu 😊

Basics of CLTK (Classical Language Toolkit) i

```
http://cltk.org/
```

```
pip install cltk
```

In the following code boxes are some example usages. There are many more tutorials on the CLTK Github (accessible through the project web page) such as Lemmatization, Creating a Lexical Disperion Plot, etc.

J-i u-v replacement i

```
# Converting J to I, V to U
from cltk.stem.latin.j_v import JVReplacer

j = JVReplacer()
j.replace('vem jam') #Out[3]: 'uem iam'
```

Declining using Collatinus i

```
from cltk.stem.latin.declension import CollatinusDecliner

decliner = CollatinusDecliner()

print(decliner.decline("via"))
decliner.decline("via", flatten=True)
```

Lemmatizer i

```
# The CLTK offers a series of lemmatizers that can be combined in a backoff
from cltk.lemmatize.latin.backoff import BackoffLatinLemmatizer

lemmatizer = BackoffLatinLemmatizer()
tokens = ['Quo', 'usque', 'tandem', 'abutere', ',', 'Catilina', ',', '']
lemmatizer.lemmatize(tokens)
```


Line Tokenization i

```
from cltk.tokenize.line import LineTokenizer

tokenizer = LineTokenizer('latin')
untokenized_text = """49. Miraris verbis nudis me scribere versus?\nHo
tokenizer.tokenize(untokenized_text)
```

Stopword Removal i

```
from nltk.tokenize.punkt import PunktLanguageVars
from nltk.stop.latin.stops import STOPS_LIST
```

```
sentence = 'Quo usque tandem abutere, Catilina, patientia nostra?'
p = PunktLanguageVars()
tokens = p.word_tokenize(sentence.lower())
[w for w in tokens if not w in STOPS_LIST]
# alternative to this Perseus list, a custom stop list can be built, s
```

Macronizer i

```
# Macronizer: Automatically mark long Latin vowels with a macron.  
# Note that the macronizer's accuracy varies depending on which tagger  
# macronized text is needed for scansion (see below)
```

```
from cltk.prosody.latin.macronizer import Macronizer  
macronizer = Macronizer('tag_ngram_123_backoff')
```

```
text = 'Quo usque tandem, O Catilina, abutere nostra patientia?'  
macronizer.macronize_text(text)  
# Out[4]: 'quō usque tandem , ō catilinā , abūtēre nostrā patientia ?'  
macronizer.macronize_tags(text) # alternatively
```

```
from cltk.prosody.latin.scanner import Scansion
from cltk.prosody.latin.clausulae_analysis import Clausulae

text = 'quō usque tandem abūtēre, Catilīna, patientiā nostrā. quam diū

s = Scansion()
c = Clausulae()

prosody = s.scan_text(text) #Out[6]: ['-uuu-uuu-u--x', 'uu-uu-uu---x']

c.clausulae_analysis(prosody)
```

Prosody Scanner i

```
# A prosody scanner is available for text which already has had its na
# The algorithm is designed only for Latin prose rhythms. It is detail
from cltk.prosody.latin.scanner import Scansion
scanner = Scansion()
text = 'quō usque tandem abūtēre, Catilīna, patientiā nostrā. quam diū
scanner.scan_text(text)
```

HexameterScanner i

```
from cltk.prosody.latin.hexameter_scanner import HexameterScanner
scanner = HexameterScanner()
scanner.scan("impulerit. Tantaene animis caelestibus irae?")
```

PentameterScanner i

```
from cltk.prosody.latin.pentameter_scanner import PentameterScanner
scanner = PentameterScanner()
scanner.scan("ex hoc ingrato gaudia amore tibi.")
# for more see http://docs.cltk.org/en/latest/latin.html
# HendecasyllableScanner, Syllabifier, etc.
```

Named Entity Recognition i

```
# There is available a simple interface to a list of Latin proper nouns
# Ergo: will recognize personal names from the list (thus you can add
from cltk.tag import ner
from cltk.stem.latin.j_v import JVReplacer

text_str = """ut Venus, ut Sirius, ut Spica, ut aliae quae primae dicuntur
jv_replacer = JVReplacer()
text_str_iu = jv_replacer.replace(text_str)

ner.tag_ner('latin', input_text=text_str_iu, output_type=list)
```


CLTK Pipeline

Corpus Import i

```
# available as tutorial Jupyter Notebooks from the CLTK Github
# https://github.com/cltk/tutorials/blob/master/2%20Import%20corpora.i
from cltk.corpus.utils.importer import CorpusImporter
my_latin_downloader = CorpusImporter('latin')
my_latin_downloader.list_corpora # show available corpora

my_latin_downloader.import_corpus('latin_text_latin_library')
my_latin_downloader.import_corpus('latin_models_cltk')

# save text in string variable
# Introduction to Cato's De agricultura
cato_agri_praef = "Est interdum praestare mercaturis rem quaerere, nis

# See http://docs.cltk.org/en/latest/latin.html#sentence-tokenization
from cltk.tokenize.sentence import TokenizeSentence
tokenizer = TokenizeSentence('latin')
```

Corpus Import ii

```
cato_sentence_tokens = tokenizer.tokenize_sentences(cato_agri_praef)
print(len(cato_sentence_tokens)) # 9

for sentence in cato_sentence_tokens:
    print(sentence)
    print()

# Import general-use word tokenizer
from cltk.tokenize.word import nltk_tokenize_words
cato_word_tokens = nltk_tokenize_words(cato_agri_praef)
print(cato_word_tokens)
cato_word_tokens_no_punt = [token for token in cato_word_tokens if token != '']
# you can remove duplicates by using the set() function on it

# There's a mistake here, though:
# capitalized words ('At', 'Est', 'Nunc') would be counted incorrectly
# So let's lowercase the input string and try again:
```

```
cato_agri_praef_lowered = cato_agri_praef.lower()  
cato_word_tokens_lowered = nltk_tokenize_words(cato_agri_praef_lowered)
```

Visualize word frequencies i

```
from collections import Counter
# You don't get the unique word forms, but count all tokens

cato_word_counts_counter = Counter(cato_cltk_word_tokens_no_punt)
print(cato_word_counts_counter)

# Lexical diversity is a simple measure of unique words divided by tot

# Lexical diversity of our little paragraph
print(len(cato_cltk_word_tokens_no_punt_unique) / len(cato_cltk_word_t
# This represents the ratio of unique words to re-reused words
```

Text reuse i

```
# Load all of Cicero's De divinatione
import os
# from your own machine (!)
div1_fp = os.path.expanduser('~/.cltk_data/latin/text/latin_text_latin_1.txt')
div2_fp = os.path.expanduser('~/.cltk_data/latin/text/latin_text_latin_2.txt')

with open(div1_fp) as fo:
    div1 = fo.read()

with open(div2_fp) as fo:
    div2 = fo.read()

# We will calculate the Levenshtein distance
# See http://docs.cltk.org/en/latest/multilingual.html#text-reuse
from cltk.text_reuse.levenshtein import Levenshtein
lev_dist = Levenshtein()
lev_dist.ratio(div1, div2)
```

```
from cltk.text_reuse.comparison import long_substring

# Aen 1.1-6
aen = """arma virumque cano, Troiae qui primus ab oris
Italiam, fato profugus, Laviniaque venit
litora, multum ille et terris iactatus et alto
vi superum saevae memorem Iunonis ob iram;
multa quoque et bello passus, dum conderet urbem,          5
inferretque deos Latio, genus unde Latinum,
Albanique patres, atque altae moenia Romae."""

# Servius 1.1
serv = """arma multi varie disserunt cur ab armis Vergilius coeperit,

print(long_substring(aen, serv))
```

Und... fröhliches Workshopen morgen! 😊

Literatur



Steven Bird, Ewan Klein und Edward Loper. *Natural Language Processing with Python*. Sebastopol, 2009.



Matthew L. Jockers. *Macroanalysis. Digital Methods and Literary History*. Chicago, 2013.



Franco Moretti. *Distant Reading*. London, 2013.



Franco Moretti. *Graphs, Maps, Trees. Abstract Models for a Literary History*. London, 2005.



Malte Rehbein. “Informationsvisualisierung”. In: *Digital Humanities. Eine Einführung*. Hrsg. von Fotis Jannidis, Hubertus Kohle und Malte Rehbein. Stuttgart: Metzler, 2017, S. 328–342.



Steffan Sinclair und Geoffrey Rockwell. “Text Analysis and Visualization: Making Meaning Count”. In: *A New Companion To Digital Humanities*. Hrsg. von Susan Schreibmann, Ray Siemens und John Unsworth. Oxford: Wiley Blackwell, 2016, S. 274–290.



Bjarne Stroustrup. *Programming. Principles and Practice Using C++*. 2nd Edition. NY: Addison-Wesley, 2014.



Jun Wu. *The Beauty of Mathematics in Computer Science*. Boca Raton: Chapman und Hall/CRC Press, 2018.